Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski jednopredmetni studij informatike

Mauro Orlić

Razvoj edukativne igre „Chroma“ u alatu Unity

Završni rad

Mentor: doc. dr. sc. Marina Ivašić Kos prof. mat. i inf.

Rijeka, Rujan 2018

.

Sadržaj

Sažetak

U ovom radu se opisuju osnove rada u alatu Unity-u i proces izrade edukativne igre „Chroma“, top-down shooter čiji je cilj naučiti igrača prepoznati pojedine RGB komponente boja.

Ukratko se opisuje sučelje alata Unity, zatim sustav komponenti na kojemu se temelji većina razvojnog procesa i nekoliko važnijih komponenata korištenih pri izradi igre. Nakon toga objašnjena je implementacija ključnih elemenata kao što su igrač, neprijatelj, sustav boja i sl. te ponašanje i dizajn istih.

1 Uvod

Izrada videoigara je bio jedan od težih poslova, pogotovo na svom početku kasnog 20. st. Tada je najveći problem bilo samo projektiranje pošto su se videoigre tek počele razvijati kao medij.

U tim danima developeri igara na raspolaganju su imali samo alate koje su si sami izradili, odnosno sve od alata do krajnje igre je bilo izrađeno *in-house*.

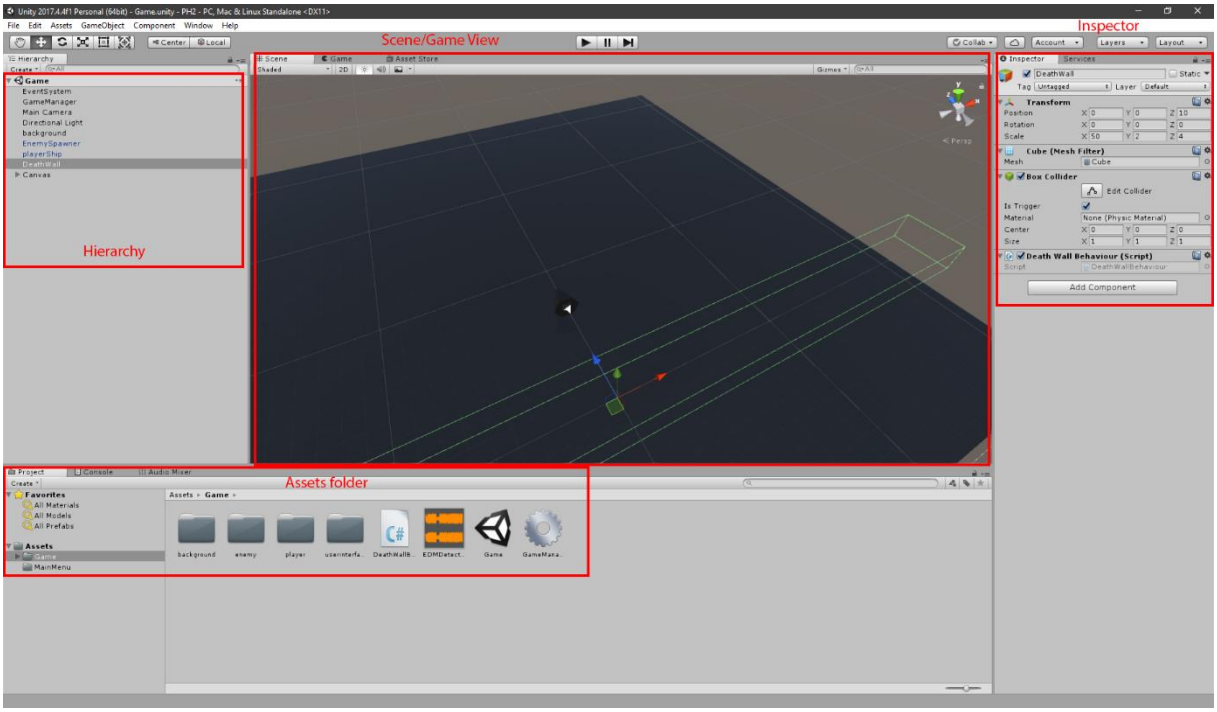
Tek u zadnjih 20-ak godina su se počeli pojavljivati unaprijed napravljeni alati koje bi studio-i videoigara mogli licencirati i koristiti za komercijalne svrhe, no tek unutar zadnjih 10 godina je takva praksa postala popularna i omogućila nastanak mnogih manjih studio-a koji ne bi imali dovoljnu inicijalnu investiciju za izradu vlastitih alata. Osim njih su se počeli pojavljivati i indie developeri; Ponekad manji timovi ali najčešće jedna osoba koja bi se samostalno bavila svim aspektima razvoja igre. Od izrade art-a, modela, animacija, zvukova do izrade programskih skripti koje upravljaju prethodno navedenim elementima.

Od svih dostupnih alata Unity ima uvjerljivo najveći broj korisnika te shodno tome naviše lako dostupnih materijala za učenje i foruma za postavljanje pitanja. Alat sam po sebi lako naučiti i koristiti, ali je također i iznimno moćan do mjere gdje ga i iznimno veliki studio-i koriste.

2 Sučelje alata Unity

Unutar Unity alata postoje mnoge opcije i podešenja pri izradi projekta. U ovom radu će biti opisani najvažniji elementi pošto je opširan i detaljan opis alata van opsega ovog rada.

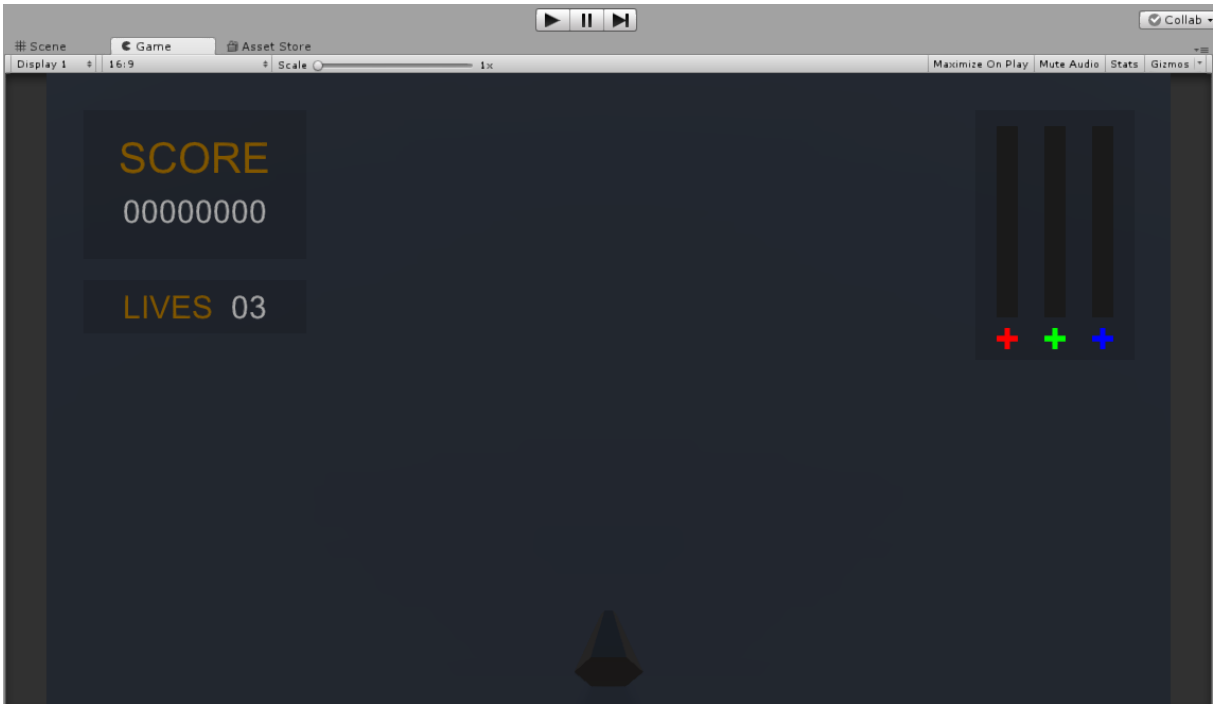
Na slici 1 vidljivi su osnovni paneli koji se najviše koriste pri izradi igre; Scene View, Game View, Hierarchy, Inspector i Asset folder.



Slika 1 Sučelje alata Unity

2.1 Game View

Game view je pogled na igru sa gledišta igrača, najčešće kroz glavnu kameru postavljenu u sceni. Pri izradi igre moguće je pokrenuti scenu u kojoj se trenutno nalazimo bez prethodnog buildanja projeta. Za vrijeme igre možemo mijenjati bilo koje njezine značajke i one će se za vrijeme izvođenje igre ažurirati.



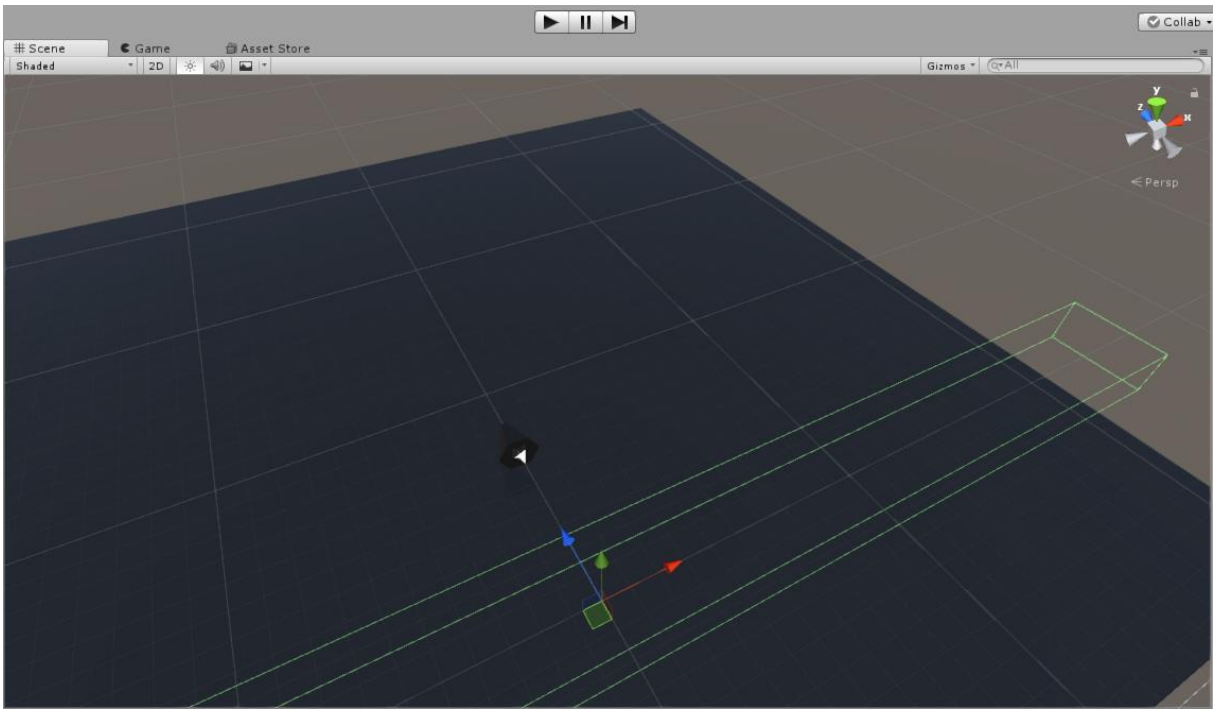
Slika 2 Game view

Na slici 2 možemo vidjeti sam prozor koji prikazuje igračevu perspektivu, pomoću Play i Pause tipke koji se nalaze pri vrhu pokrećemo igru unutar Unity editora. Važno je napomenuti da će performansi biti grozni čak i na iznimno dobrom hardveru pošto elementi igre nisu „baked-in[1]“. Svrha pokretanja igre u ovom stanju je debugiranje i brzo testiranje pošto bi buildanje cijelog projekta za svaki mali test tražilo previše vremena.

[1] Radi povećanja performansa kod programa, dinamničnim u prirodi elementima koji su u statičnoj ulozi se unaprijed “bake-aju”, odnosno neke od varijabli elemenata postanu konstante te se njihov izračun preskače.

2.2 Scene View

Na slici 3 prikazan je Scene View. Unutar njega možemo direkno manipulirati poziciju, rotaciju i skalu objekata. Svrha Scene View-a jest davanje mogućnosti developeru da direktno, vizualno može namještati značajke prisutnih objekata koje nisu nužno primjetljive unutar Game View-a.



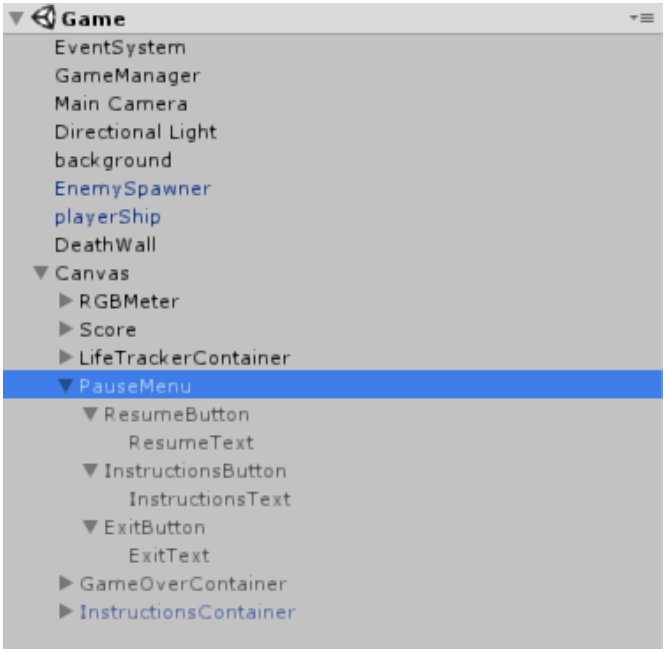
Slika 3 Scene view

Kamera kroz koju developer vidi u Scene View-u nije dio igre već je dio samog alata koju je moguće slobodno kontrolirati mišem i tipkovnicom. Scene View je dostupan i moguće je interaktirati sa objektima unutar scene-a ne samo dok je igra statična unutar editora nego i dok je ona pokrenuta(naravno, unutar editora).

2.3 Hierarchy

Na slici 4 prikazana je hijerarhija gameObject[2]-a. Pri pokretanju scene objekti se pokreću redom od vrha prema dnu što može biti važno ukoliko imamo nekakav slijed operacija između objekata.

Također možemo uspostavljati veze tipa parent-child između objekata. Efekt toga je da child objekti za referencu svoje pozicije ne koriste apsolutno ishodište scene već koriste



Slika 4 Hierarchy

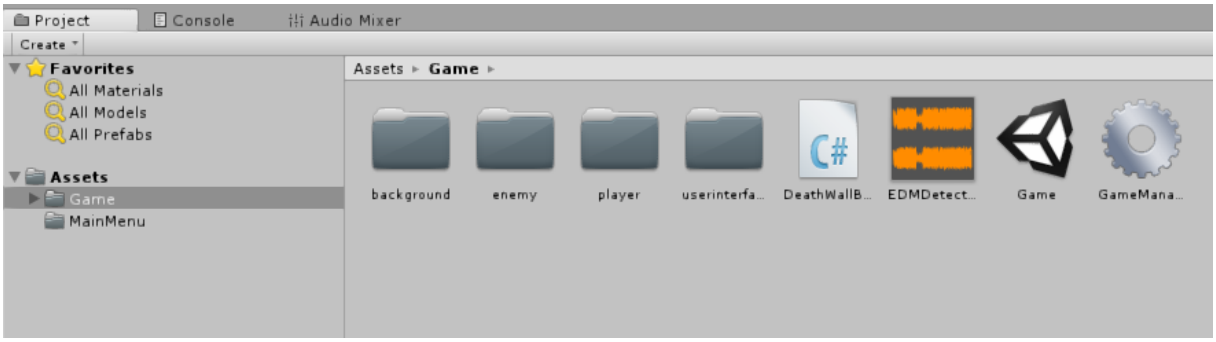
trenutnu poziciju svog roditelja. Također child objekti poprimaju stanje „aktiviranosti“ od svog roditelja, odnosno ako je unutar scene parent objekt disable-an(nije vidljiv, statičan je, ne obavlja ikakve radnje) biti će i sva njegova djeca.

Npr. ako imamo objekt PauseMenu koji ima kao child objekte tipke Resume, Options, Exit možemo prikazati ili sakriti cijeli meni aktivacijom ili deaktivacijom samo jednog objekta; PauseMenu-a.

[2] GameObject-i su svi objekti koji se nalaze u sceni, I to ne objekti u fizičkom, već programskom smislu pošto ne moraju imati tijelo. GameObject ovisno o postavkama može imati ulogu neprijatelja, kamere, izvora svjetla…

2.4 Assets folder

Na slici 5 vidimo prikaz Assets foldera. Mapa u koju ručno postavljamo sve datoteke koje želimo koristiti u svom projektu. Asseti se mogu podijeliti na eksterne i interne s obzirom na Unity editor.



Slika 5 Assets

Eksterni assetovi mogu biti slike, spriteovi, zvukovi, glazba, 3D modeli, animacije, meshevi. Moguće je importanje datoteka iz drugih programa kao npr. .psd datoteke photoshopa ako nam izvezena slika nije dovoljna.

Interni assetovi su većinom oni koje sam Unity editor može stvoriti i izmjenjivati. Oni mogu biti bilo što, od materijala objekata, shadera, do gotovih karaktera s uključenim rigom, skriptama i sl.

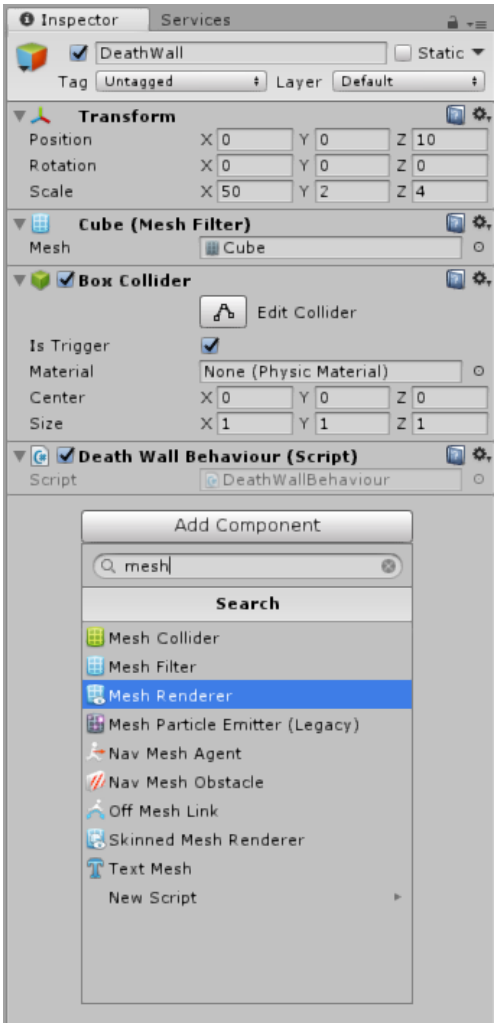
2.5 Inspector

Na slici 6 sa desne strane je prikaz inspektora gameObject-a. Ovdje možemo mijenjati značajke komponenata bilo kojeg gameObject-a koji je u sceni. Bazično, novostvoreni objekt dolazi samo sa transform komponentom te je potrebno dodavati na njega druge komponente kako bi on mogao obavljati neku funkciju.

Npr. na slici vidimo DeathWall, to je objekt pri dnu koji detektira neprijatelje koji su prošli dno ekrana. On ih uništava te igraču oduzima život.

Pošto treba detektirati objekte koji mu dođu u prostor prvo treba imati svoj prostor, odnosno treba mu mesh. Nakon toga treba pomoću tog mesh-a detektirati kolizije, pa na njega stavimo Collider, i naposljetku dodajemo komponentu skriptu koja kontrolira njegovo ponašanje.

U sličnom stilu se izrađuju svi gameObject-i.



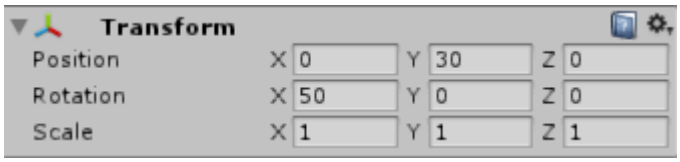
Slika 6 Inspector

3 Sustav komponenti

U ovom poglavlju će biti pojašnjene najvažnije komponente korištene u igri „Chroma“ mada ih ima mnogo više.

3.1 Transform

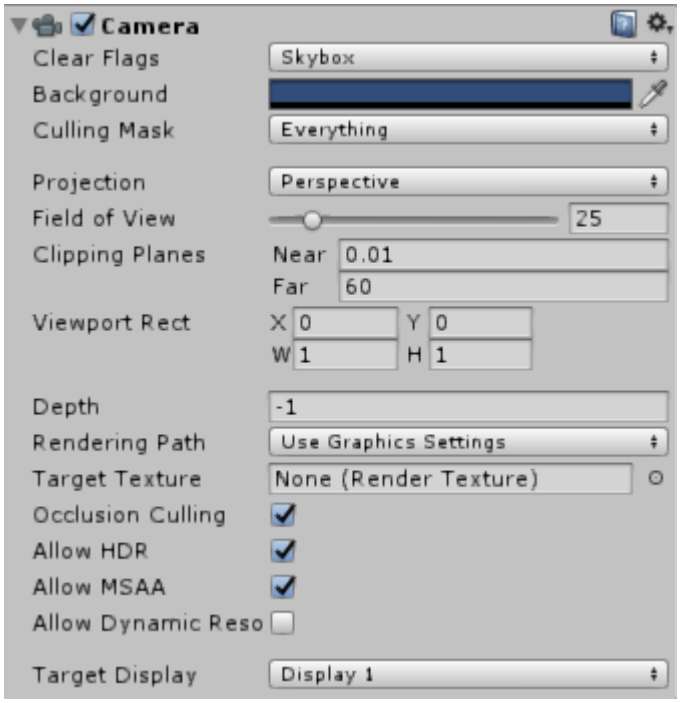
Transform je najosnovnija komponenta koju svaki objekt mora imati. Pomoću nje definiramo poziciju i rotaciju gameObject-a te ga je moguće skalirati(vrijednost skale se primjenjuje prije rotacije, t.d. ako neki rotirani objekt skalira on će se skalirati kao da su mu vrijednosti rotacije 0)



Slika 7 Transform

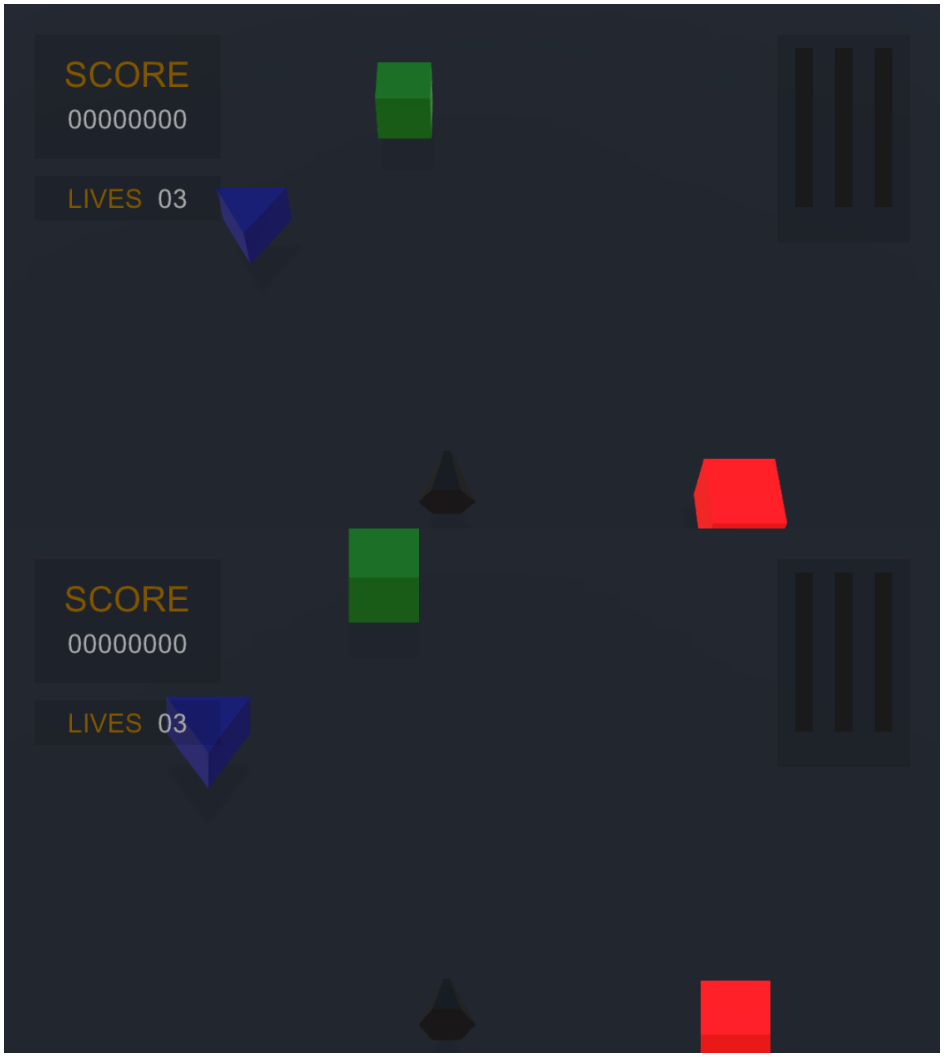
3.2 Kamera

Kamera je komponenta kroz koju igrač percipira igru.



Slika 8 Camera

Ovisno o tome kakvu igru izrađujemo možemo imati kameru čija je projekcija perspektivna[3] ili ortografijska[4].



Slika 9 Perspektivna(gore) i Ortografska(dolje) projekcija

Za performanse igre također su važni Clipping Planeovi. Od ishodišta kamere do udaljenosti „Near“ kamera neće renderati išta, to također vrijedi nakon udaljenosti „Far“ plane-a.

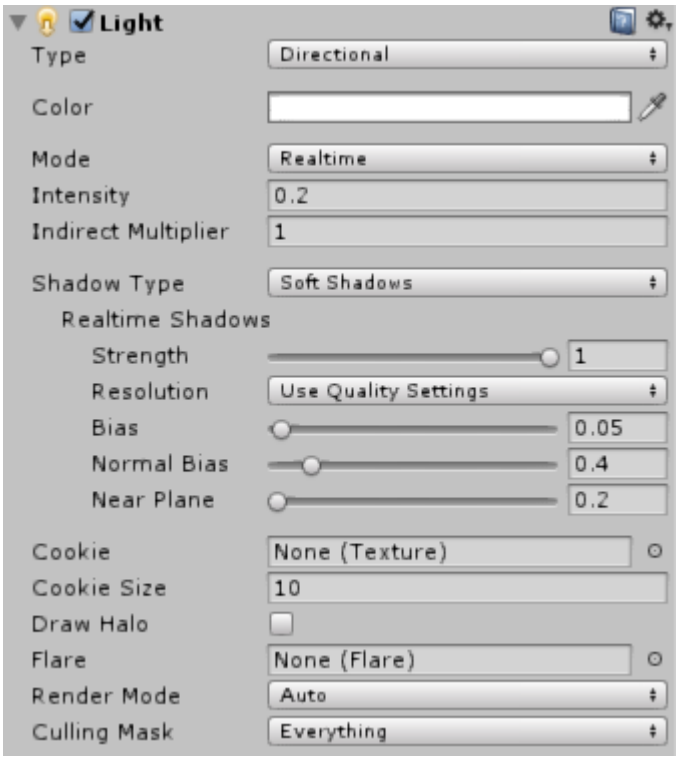
Ovo je vrlo korisno ukoliko imamo vrlo veliki svijet unutar jedne scene te bi renderanje cijele scene odjednom bilo previše zahtjevno. Također povećavanjem „Near“ udaljenosti možemo spriječiti flickeranje objekata koji se previše približe kameri.

[3] Iz točke ishodišta kamere se projicira pravokutnik pod određenim vertikalnim I horizontalnim kutem, blisko realističnoj percepciji prostora.

[4] Projekcija je oblika kvadrata od ishodišta do kraja projekcije, pogodno za 2D igre, ili 3D igre treće perspektive u kojima je rotacija kamere fiksna.

3.3 Svjetla

Osvjetljenje je esencijalni dio bilo koje igre, od osnovnog omogućavanja igraču da zapravo percipira svoje okruženje do složenih uređenja svjetala da stvore neki ugođaj.



Slika 10 Light

Osim boje i intenziteta svjetla možemo postaviti tip svjetla. Dva osnovna tipa svjetla koji su dostupni su Directional i Point.

Directional osvjetljenje „baca“ paralelne zrake na sve objekte u sceni jednakim intenzitetom te se najčešće koristi za simulaciju sunčevog svjetla.

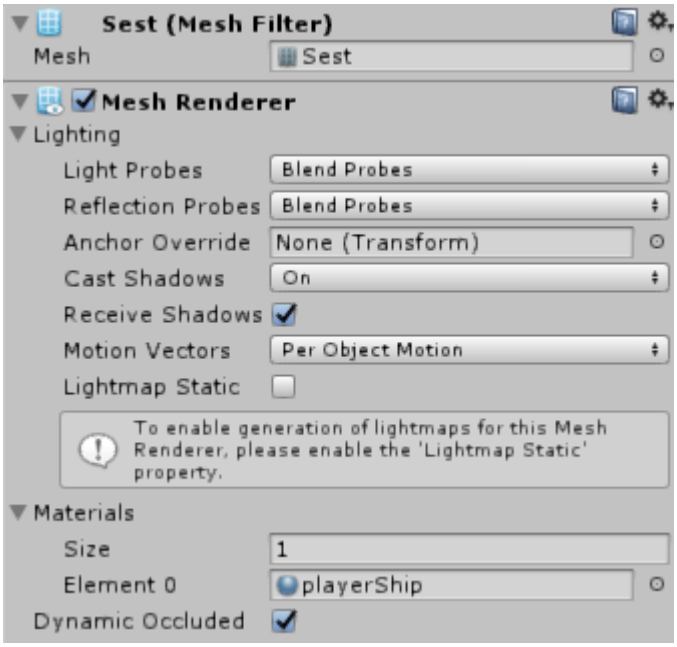
Point osvjetljenje ima ishodište u jednoj točki, osvjetljava sve oko sebe te mu jačina opada s udaljenosti od ishodišta.

Također korisna opcija ako želimo poboljšati performanse igre jest odabir između Realtime i Baked modova svjetla. Baked mod unaprijed rendera razine osvjetljenja i sjene objekata. Ovo je vrlo korisno ako imamo ogromno statično okruženje, pogotovo ako se sastoji od mnogo poligona pošto se osvjetljenje izračuna jednom. Ovo očito neće funkcionirati dobro kod dinamičnih objekata koji se kreću.

Kod Realtime osvjetljenja ponovno se izračunavaju osvjetljenje i sjena za svaki objekt, svaki frame što je mnogo zahtjevnije od Baked moda ali daje realističnije osvjetljenje.

3.4 Mesh i Mesh Renderer

Da bi objekt imao svoj oblik, on prvo mora imati svoj mesh, odnosno tijelo prikazano mnoštvom povezanih poligona. Npr. unutar igre Chroma imamo gameObject playerShip(brod koji igrač kontrolira). Njegov mesh je šesterostrana prizma, odnosno vrhovi i bridovi koji ga čine.



Slika 11 Mesh, Mesh Renderer

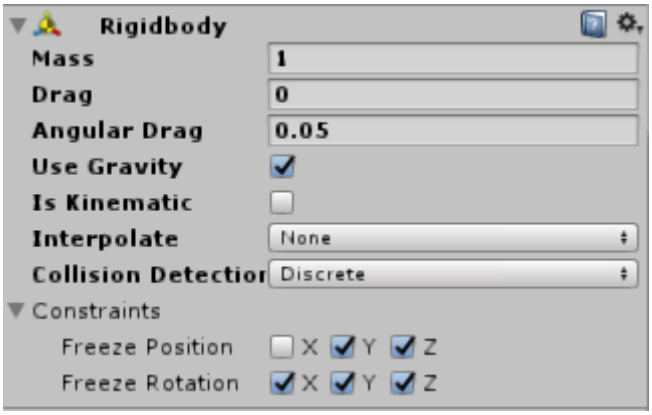
Pri importanju modela iz programa za izradu 3D modela(Blender, Maya, Houdini...) komponenta Mesh Filter reducira model na svoj bazični mesh te ga dalje prosljeđuje Mesh Renderer-u zaduženom za renderanje mesh-a.

Renderer koristi Light Probe-ove za detektiranje razine svjetla u direktnoj okolini da bi mogao odrediti svjetlinu materijala mesh-a za svaku točku mesh-a.

Mesh-u se može nadodati još mnogo vizualnih efekata kao poprimanje svjetline sa svjetlosti reflektirane s nekog drugog objekta, bacanje sjene te još mnogo.

3.5 Rigidbody

Rigidbody komponenta daje gameObject-u fizička svojstva, utjecanje gravitacije na objekt, postavljanje vektora brzine i sile pomoću skripta.



Slika 12 Rigidbody

3.6 Collider

Collider komponenta diktira kako će se gameObject ponašati pri koliziji sa drugim objektima. Osnovna svrha Collidera jest fizička interakcija između objekata, također je moguće odabrati materijal koji se koristi pri koliziji (npr. objekti gumenog materijala će pri koliziji odskakati više od objekata metalnog materijala).

Slika 13 Collider

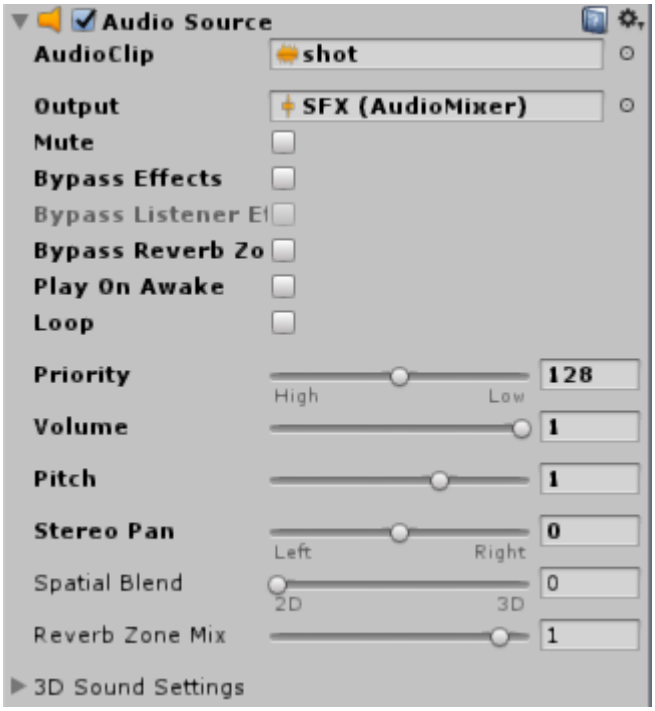
Pošto je detekcija kolizije dvaju tijela računalno zahtjevno, za većinu objekata složenijih mesh-eva se uzima jednostavno geometrijsko tijelo kao aproksimacija koja je u većini slučajeva više nego adekvatna(npr. pri detekciji kolizije igrača sa zidom u nekom FPS-u nećemo koristiti kompleksan model čovjeka nego oblik uspravne kapsule). Ukoliko svejedno želimo da gameObject koristi mesh za kolizije koristimo Mesh Collider.

Ukoliko upalimo opciju „Is Trigger“ gameObject gubi fizičku prirodu kolizije te nam omogućuje da kolizije detektiramo i kontroliramo pomoću skripta. Primjer svrhe toga bi bila situacija gdje imamo objekte igrača i neprijatelja gdje pri koliziji (dodiru) neprijatelj radi štetu igraču.

3.7 Audio Source

Glazba, zvučni efekti, postavljaju ugođaj igri, daju dodatni osjet radnjama koje igrač vrši ili doživljava(udarac neprijatelja oružjem, zvuk koraka po travi, zvuk koraka po kamenu, zvučni efekt koji obavještava igrača).

gameObject može sadržavati više Audio Source komponenta te ih puštati po potrebi. Radi olakšanja uporabe zvukova bez dodatnog skriptiranja moguće je postaviti da se zvuk reporoducira pri stvaranju objekta(npr. pri ispaljivanju metka) ili da se automatski ponavlja pri završetku reprodukcije(glazba glavnog menija).



Slika 14 Audio Source

Važno je napomenuti da se jedan Audio Source na jednom objektu ne može reproducirati više puta odjednom, već ako zvuk tijekom izvođenja ponovno dobije poziv da se reproducira nećemo čuti više instanca istog zvuka već će se zvuk zaustaviti i ponovno repoducirati. Također je važno napomenuti da ako je zvuk u tijeku izvođenja te njegov gameObject roditelj biva uništen, zvuk će prestati sa reprodukcijom(važno kod neprijatelja koji imaju zvučni efekt umiranja i sl.).

Audio Source komponenta se ponaša kao realističan izvor zvuka, s povećanjem udaljenost od Audio Source-a jačina zvuka slabi, pri prolasku pokraj objekta sa Audio Source-om se primjenjuje Dopplerov efekt.

Prije nego što zvuk dospije do igračevih slušalica moguće ga je proslijediti Audio Mixeru koji dalje primjenjuje efekte na njega. Pomoću Audio Mixera moguće je grupirati srodne Audio Source-ove umjesto da se na svaki zvuk ručno postavljaju efekti(npr. da bi smo lakše kontrolirali glasnoću zvučnih efekata staviti ćemo sve zvučne efekte u „SFX“ grupu unutar AudioMixer-a te kontrolirati glasnoću „SFX“ grupe).

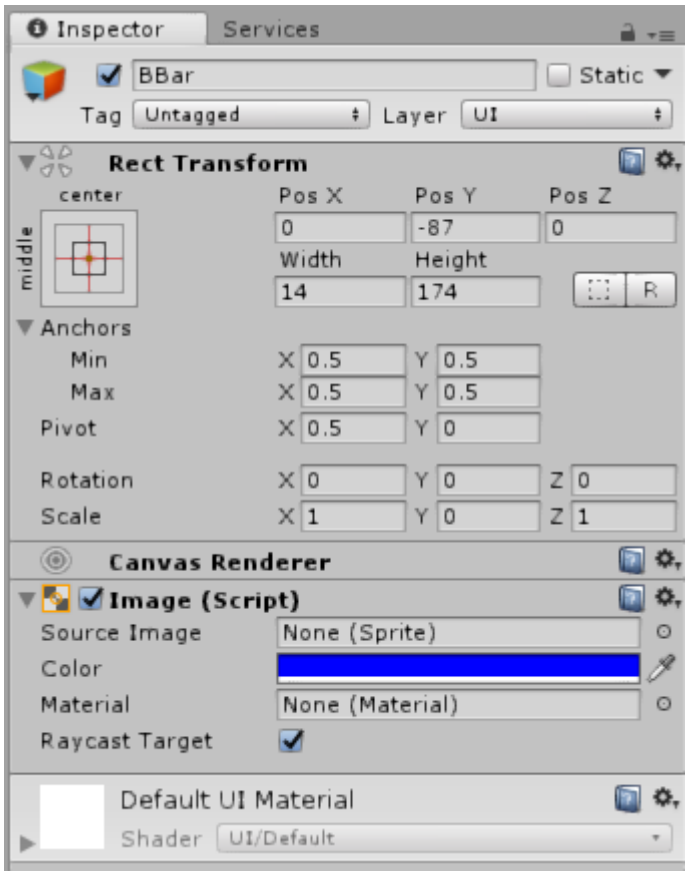
3.8 Audio Listener

Iznimno jednostavna komponenta koja sluša za sve zvukove sa pozicije gameObject-a na kojem se nalazi.

Slika 15 Audio Listener

3.9 User interface komponente

Postoji mnoštvo komponenta koje su potrebne ovisno o tome što želimo prikazati igraču u user interface-u te opisivanje svih bi bilo van opsega ovog rada. Umjesto toga biti će opisani samo nekoliko osnovnih.



Slika 16 UI komponente

Canvas komponenta čini objekt na kojem se nalazi platnom za UI(User Interface) komponente. Svi objekti koji su dijelovi UI-a se postavljaju kao child Canvas-a. Canvas je moguće prikazivati kao overlay kameri(Canvas se „zalijepi“ preko slike koju kamera snima), može ga se prikazivati kao da je na određenoj udaljenosti od kamere pod nekim kutem ali i dalje se ponaša kao overlay, ili ga je moguće prikazivati kao normalan objekt u sceni(na njega utječe osvjetljenje, renderani objekti mogu blokirati pogled na njega i sl.).

RectTransform komponenta je slična običnoj Transform komponenti samo što RectTransform funkcionira na 2D površini umjesto 3D prostoru. Ukoliko parent objekt(2.3) također sadrži RectTransform komponentu možemo postaviti anchor child objekta relativno prema parent objektu(npr. da neovisno o veličini i poziciji parent objekta child objekt se

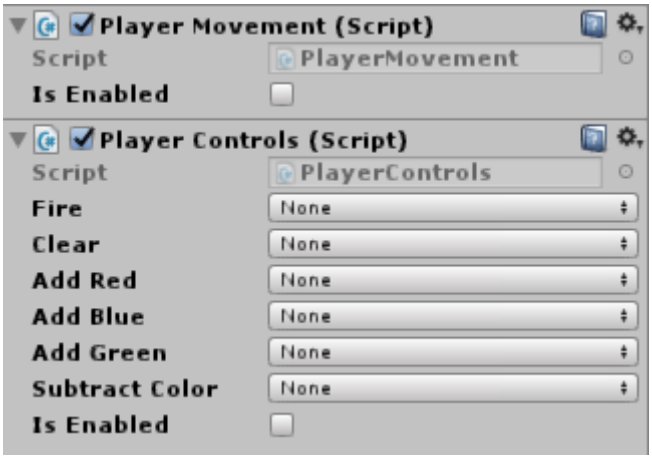
uvijek nalazi u gornjem lijevom kutu parent objekta). Pivot objekta je točka objekta oko koje se vrše operacije rotiranja i skaliranja.

Graphic Raycaster – jednostavnim riječima, „baca“ zraku iz trenutne pozicije miša na Canvas, služi kao baza za interakciju sa Canvasom unutar igre putem miša(klikanje, držanje i povlačenje UI elementa)

Među ostale komponente za UI spadaju tipke, slideri, checkboxovi, dropdown menuovi, slike, scrollbarovi i sl.

3.10 Skripte

Kada je potrebno nekakvo specifično ponašanje od strane gameObjekata kao jedina opcija ostaje izrada vlastitih komponenti. To se izvodi na način da developer samostalno napiše program koji će se izvršavati.



Slika 17 Skripte

Dolje vidimo primjer izgenerirane skripte. Standardno dolazi sa Start() i Update() metodama. Start() metoda se izvršava pri inicijalizaciji gameObject-a, a Update() pri svakom slijedećem renderanom frame-u.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class newScriptComponent : MonoBehaviour {

    void Start () {

    }

    void Update () {

    }
}
```

4 Izrada igre „Chroma“

4.1 RGB sustav boja

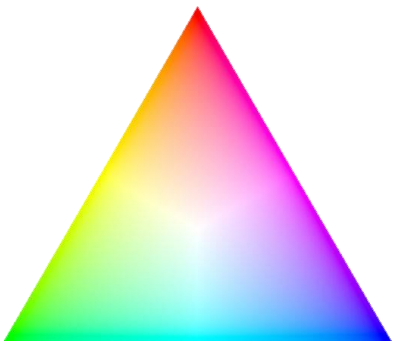
4.1.1 Suptraktivni i aditivni modeli boja

Kod suptraktivnih modela imamo bijelu svjetlost koja se reflektira sa bijele površine obojane nekim elementarnim tintama. Svaka tinta sprječava reflektiranje jedne boje. Preklapanjem tinti blokira se unija boja koju svaka od tinti blokira. Reflektirane zrake sa te površine su boje dobivene suptraktivnim modelom boja. Koristi se kod bojanja objekata koji se oslanjaju na strani izvor svjetlosti za svoju vidljivost(isprintani tekts, posteri, prometni znakovi, drveća...).

Kod aditivnog modela imamo crnu površinu koju osvjetljavamo zrakama pojedinih elementarnih boja, kombiniranjem njihovih zraka te elementarne boje se „zbrajaju“ i miješaju. Aditivni modeli se koriste kod svih objekata koji produciraju svjetlo u svrhu percepcije boje(mobiteli, monitori, svjetla, LED paneli).

4.1.2 RGB model

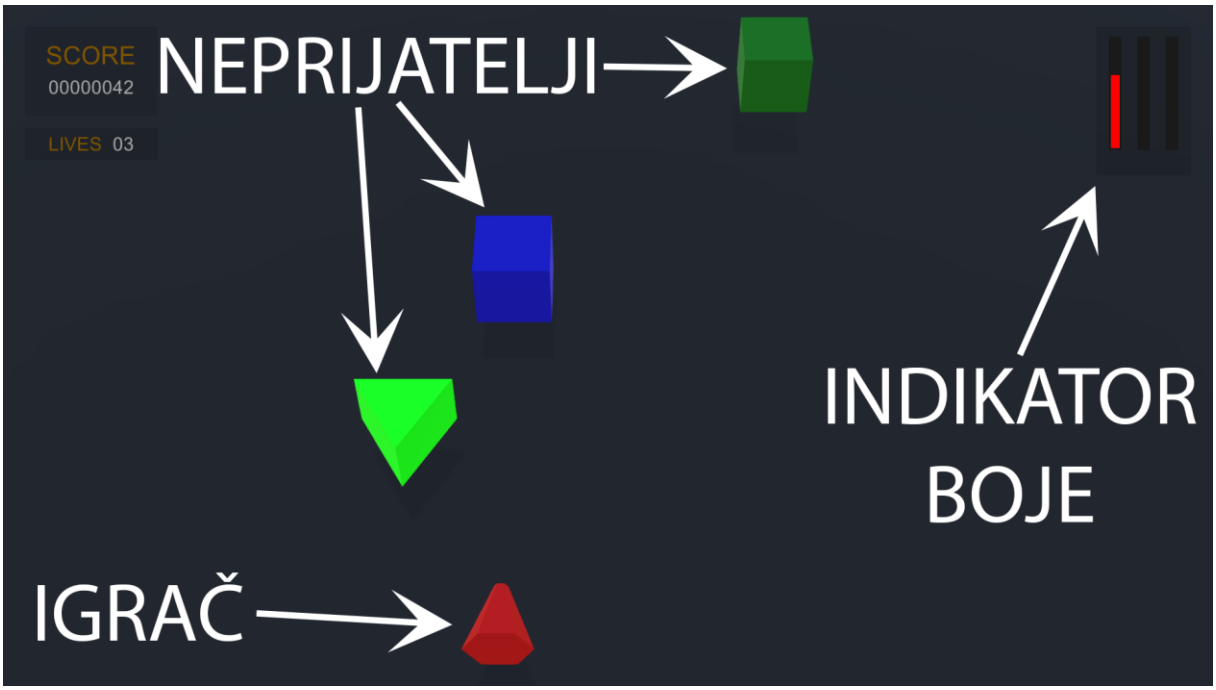
RGB je aditivni model boja koji koristi uređenu trojku za jednoznačno određivanje boja. Članovi te trojke su redom crvena, zelena i plava. Odabran je kao objekt učenja pošto se najčešće koristi. Na desnoj strani možete vidjeti prikaz RGB sustava boja, u vrhovima trokuta se nalaze elementarne boje, svaka druga točka unutar trokuta je dobivena miješanjem boja.



Slika 18 RGB trokut

4.2 Dizajn igre

Za igru je odlučeno da će educirati igrača o aditivnosti boja unutar RGB modela boja. Osmišljena je kao endless shooter sa progresivnom težinom. Igrač kontrolira svoj spaceship pri dnu ekrana, cilj mu je uništiti nadolazeće neprijatelje prije nego što dođu do kraja ekrana.



Slika 19 Game

Igrač ima na raspolaganju tri elementarne boje, crvenu, plavu i zelenu, te za svaku po 4 stupnja intenziteta: 0%, 33%, 66% i 100%. Svaki nadolazeći neprijatelj je obojan nekom bojom koju igrač mora sastaviti zbrajanjem elementarnih boja koje su mu na raspolaganju.

4.3 Objekt „Manager“

Nevidljivi gameObject koji persistira između loadanja scena i samo sadrži settings-e koje drugi objekti isčitavaju.

4.4 Klasa „ColorRGB“

Pošto u Unity gameEngine-u ne postoji klasa koja podržava konstrukciju boja sa float ili int vrijednostima od 0 do 255, niti podržava zbrajanje boja bilo je potrebno napraviti vlastitu klasu sa pripadnim konstruktorom i overloadanim operatorima za potrebe igre.

4.5 Player character

Igrač kontrolira playerShip pomoću miša i tipkovnice, može se micati lijevo-desno pomoću miša te pomoću tipkovnice dodaje boje na playerShip. Igrač mišem također ispaljuje metak odabrane boje ili resetira boju ukoliko mu ne odgovara.

Kao što se može vidjeti u donjem dijelu koda pri ispaljivanju metka stvori se nova instanca metka na poziciji playerShip-a te poprima njegovu boju.

```
        public void FireBullet()
        {
            GameObject newBullet = Instantiate(bullet, new Vector3(transform.position.x, bulletSpawnY,
    bulletSpawnZ), bullet.transform.rotation);
            newBullet.GetComponent<BulletBehaviour>().bulletColor = playerColor;
            newBullet.GetComponent<BulletBehaviour>().RGBMeterBehaviour =
    RGBMeter.GetComponent<RGBMeterBehaviour>();
        }
```

4.5.1 Kretanje

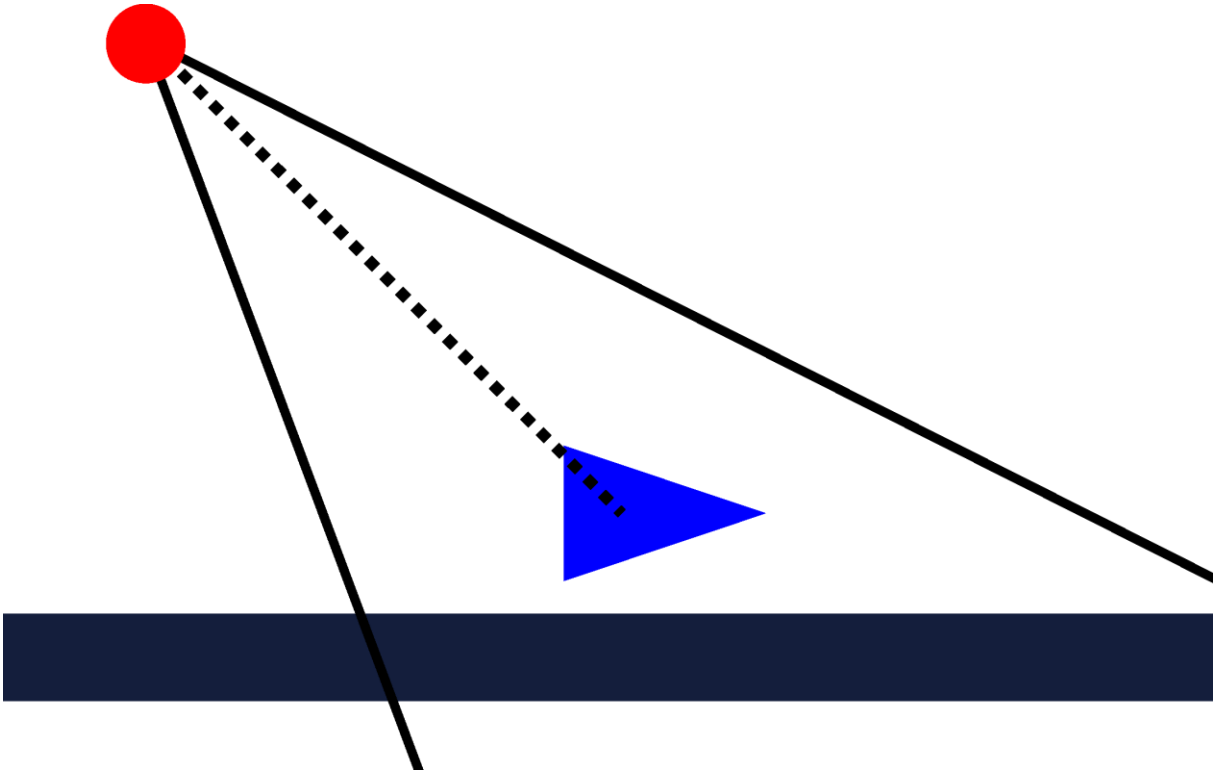
Kretanje igrača se vrši kretnjom miša, to je realizirano pomoću PlayerMovement.cs komponente. Glavni problem je kako translatirati 2D poziciju miša na ekranu na 3D poziciju igračevog playerShip-a. To je realizirano pomoću slijedećeg koda.

```
    void Start()
    {
        isEnabled = true;
        playerY = 0f;
        playerZ = 16.5f;
        playerShipDistanceToScreen = Camera.main.WorldToScreenPoint(gameObject.transform.position).z;
        transform.position = new Vector3(0f, playerY, playerZ);
    }

    void Update()
    {
        if (isEnabled)
        {
            mousePosition = new Vector3(Input.mousePosition.x, Input.mousePosition.y,
playerShipDistanceToScreen);
            transform.position = new Vector3(Camera.main.ScreenToWorldPoint(mousePosition).x, playerY,
playerZ);
        }
    }
```

Pošto se playerShip kreće isključivo lijevo-desno(po X osi) ostale smjerove postavljamo kao konstante pošto se neće mijenjati, osi Y i Z.

Jedini problem koji ostaje je izračun X koordinate. Moguće je pomoću metode ScreenToWorldPoint dobiti poziciju unutar 3D gameWorld-a pomoću X i Y pozicija miša na 2D prostoru ekrana, no rezultatna pozicija bi onda bila pravac pošto 2D prostor nema Z komponentu. Shodno tome mi moramo sami definirati na kojoj udaljenosti od kamere(Z komponenta) uzimamo X i Y komponente.



Slika 20 Bokocrt kamere i playerShip-a

Pošto je cilj kontrolirati playerShip kao Z komponentu uzet ćemo udaljenost između kamere i playerShip-a. To radimo na način da stvorimo vektor između njih te uzmemo Z komponentu tog vektora, tako dobivamo vrijednost varijable playerShipDistanceToScreen.

Uzimamo novi vektor mousePosition , kao X komponentu uzimamo X poziciju miša na ekranu, Y komponenta miša je irrelaventna pošto se playerShip po njoj neće kretati, za Z komponentu uzimamo prethodno izračunatu varijablu playerShipDistanceToScreen. Proslijeđivanjem tog vektora metodi ScreenToWorldPoint dobiti ćemo poziciju te točke u koordinatnom sustavu gameWorld-a.

Pomoću keyword-a „transform“ pristupamo transform komponenti gameObject-a na kojem se skripta nalazi(u ovom slučaju gameObject playerShip) i mijenjamo njegovu poziciju na novi vektor gdje je X komponenta uzeta iz vektora dobivenog metodom ScreenToWorldPoint, a Y i Z komponente su konstante koje smo definirali na početku.

4.5.2 Kombiniranje i ispaljivanje boja

Komponenta PlayerControls.cs upravlja kontrolama playerShip-a koji se odnose na tipkovnicu, radi jednostavnosti biti će prikazan samo jedan dio koda dovoljan da se objasni način na koji skripta funkcionira.

```
public class PlayerControls : MonoBehaviour
{
    GameObject Manager;
    GameObject RGBMeter;
    public KeyCode Fire;
    public KeyCode AddRed;
    public KeyCode SubtractColor;
    ColorRGB playerColor;
    public bool isEnabled;
    void Start()
    {
        isEnabled = true;
        Manager = GameObject.Find("Manager");
        RGBMeter = GameObject.Find("RGBMeter");

        Fire = Manager.GetComponent<Manager>().Fire;
        AddRed = Manager.GetComponent<Manager>().AddRed;
        SubtractColor = Manager.GetComponent<Manager>().SubtractColor;

        playerColor = new ColorRGB(0, 0, 0);
    }
```

Prvo iz persistentnog Manager(4.3) objekta dohvaćamo KeyCode-ove koji se koriste za pojedine akcije.

```
void Update()
{
    if (isEnabled)
    {
        if (Input.GetKeyDown(Fire))
        {
            FireBullet();
            UpdateLastColor();
            playerColor = new ColorRGB(0, 0, 0);
            UpdateCurrentColor();
        }

        if (Input.GetKeyDown(AddRed))
        {
            if (Input.GetKey(SubtractColor))
            {
                playerColor -= new ColorRGB(85, 0, 0);
            }
            else
            {
                playerColor += new ColorRGB(85, 0, 0);
            }
            UpdateCurrentColor();
        }
    }
}
```

Pri svakom frame-u se provjerava je li započeto pritiskanje svake od tipki pomoću metode Input.GetKeyDown(). Moguće je također koristiti Input.GetKey() no ona provjerava svaki frame je li tipka pritisnuta te bi akcija vezana za nju bila izvršena svaki frame za koji je ta tipka bila pritisnuta. Ukoliko želimo izvršiti neku akciju pomoću više tipki preporučljivo je da jedna tipka koristi GetKeyDown() metodu a sve ostale GetKey() metodu. Razlog tome je da smo koristili GetKeyDown() na svim tipkama morali bi smo pritisnuti sve potrebne tipke unutar istog frame-a ali na prethodno opisan način bi prvo GetKey() tipke bile pritisnute te bi one time omogućile jednu aktivaciju pomoću tipke GetKeyDown().

Ukoliko stisnemo tipku za ispaljivanje metka poziv za ispaljivanje će se proslijediti glavnoj komponenti koja kontrolira igrača, te će se ispaljena boja ažurirati kao prethodno ispaljena boja u RGBMeter-u. Nnakon toga boja playerShip-a se resetirata na crnu.

Ukoliko stisnemo jednu od tipki za dodavanje boja izvršiti se dodavanje jednog stupnja elementarne boje na koju se tipka odnosi te će se ažurirati boja playerShipa.

```
    private void UpdateCurrentColor()
    {
        gameObject.GetComponent<PlayerController>().playerColor = playerColor;
        gameObject.GetComponent<PlayerController>().UpdatePlayerColor();
        RGBMeter.GetComponent<RGBMeterBehaviour>().UpdateCurrentColor();
    }

    private void FireBullet()
    {
        gameObject.GetComponent<PlayerController>().FireBullet();
    }
    private void UpdateLastColor()
    {
        RGBMeter.GetComponent<RGBMeterBehaviour>().UpdateLastColor();
    }
}
```

Pošto PlayerControls.cs je isključivo namijenjen handle-anju inputa sa tipkovnice ispaljivanje metka i ažuriranje poslijednje boje RGBMeter-a se ne izvršavaju u njemu već se pozivi metoda proslijede pripadnim komponentama.

4.6 Spawner neprijatelja

EnemySpawner objekt je prazan gameObject koji ima samo EnemySpawner.cs skriptu vezanu za sebe, njegov posao je instanciranje neprijatelja i podešavanje težine igre s vremenom.

4.6.1 Odabir točke spawnanja

Pri instanciranju neprijatelja njihova inicijalna pozicija je slična poziciji playerShip-a u smislu da imaju konstantnu Y i Z komponentu te varijabilnu X komponentu, ovaj dio koda se bavi generiranjem te X komponente.

```
spawnPoints = new float[] { -7.5f, -5f, -2.5f, 0f, 2.5f, 5f, 7.5f };
lastSpawnPoint = 4;

float getSpawnPoint()
{
    int nextSpawnPoint = Random.Range(0, spawnPoints.Length);

    if (nextSpawnPoint == lastSpawnPoint)
    {
        if (nextSpawnPoint == 0)
        {
            nextSpawnPoint = 1;
        }
        else
        {
            nextSpawnPoint--;
        }
    }
    lastSpawnPoint = nextSpawnPoint;
    return spawnPoints[nextSpawnPoint];
}
```

Potencijalne točke u kojem se neprijatelji mogu pojaviti(spawnPoint-ovi) su definirani kao skup točaka. U metodi getSpawnPoint() se nasumično odabire indeks jedne točke te metoda vraća vrijednost na tom indeksu.

U slučaju gdje se isti spawnPoint odabere dva puta za redom postoji vjerojatnost ovisno o težini igre da će se neprijatelji preklapati. U svrhu rješavanja tog problema uvodimo varijablu lastSpawnPoint koja prati indeks posljednje odabrane točke. Nakon generacije trenutne točke ukoliko je ona jednaka prethodnoj odabire se neka druga točka

4.6.2 Sustav dinamične težine

Nije pošteno novom igraču od početka igre davati kompleksne neprijatelje najvećom brzinom te je u svrhu rješavanja tog problema uveden je sustav koji povećava težinu igre s vremenom te također ovisno o sposobnosti igrača.

```
void Start()
{
    InvokeRepeating("increaseMovementSpeed", 1600f, 10f);
    InvokeRepeating("decreaseSpawnDelay", 10f, 10f);
    InvokeRepeating("increaseColorComplexity", 1f, 1f);
}
void increaseMovementSpeed()
{
    currentMovementSpeed = Mathf.Min(maxMovementSpeed, currentMovementSpeed + 0.01f);
}
void decreaseSpawnDelay()
{
    currentSpawnDelay = Mathf.Max(minSpawnDelay, currentSpawnDelay * 0.995f);
}
void increaseColorComplexity()
{
    currentColorComplexity = Mathf.Min(maxColorComplexity, currentColorComplexity + 1);
}
public void CorrectHit()
{
    currentColorComplexity += killStreak;
    killStreak = Mathf.Min(50, killStreak + 1);
}
public void IncorrectHit()
{
    killStreak /= 2;
}
```

Parametri po kojima se definira težina igre su slijedeći: brzina kretanja neprijatelja(currentMovementSpeed) određuje koliko vremena igrač ima za uništiti neprijatelja nakon što se pojavi na ekranu, interval između pojavljivanja neprijatelja(currentSpawnDelay) otežava igru na način da igrač mora istovremeno pratiti više neprijatelja i čini igru dinamičnijom i zanimljivijom od samog spuštanja jednog po jednog neprijatelja. Naposljetku vidimo varijablu currentColorComplexity što se koristi kao apstraktna mjera za težinu prepoznavanja boje.

Pomoću metode InvokeRepeating() možemo povećavati težinu igre u intervalima po želji. Prvi argument InvokeRepeating() prima naziv metode koju treba ponavljati u obliku string-a, drugi argument je broj sekundi nakon kojih će se metoda početi ponavljati a treći argument je dužina intervala ponavljanja u sekundama.

Kompleksnost boje se osim svojih bazičnih 1 po sekundi može povećavati brže ovisno o sposobnosti igrača tj. što više igrač vrši točne pogotke zaredom to će brže igrač napredovati kroz igru.

4.6.3 Odabir boja neprijatelja

Pri odabiru boje neprijatelja treba uzeti u obzir težinu igre, odnosno da težina igre reflektira težinu prepoznavanja komponenata boja. Za tu svrhu uvedeno je pet klasa boja progresivne težine prepoznavanja:

1. Sve tri elementarne boje(crvena, zelena, plava) se postavljaju na 0. Jedna od njih se nasumično izabire te joj se nasumično izabire vrijednost iz skupa vrijednost {0, 85, 170 ili 255}. Generirane boje su tonovi crvene, zelene ili plave.
2. Sve tri elementarne boje se postavljaju na istu nasumično izabranu vrijednost te se jedna od njih tri nasumično postavi na 0. Generirane boje su tonovi cijan, pupurnocrvene, i žute boje.
3. Svaka od tri elementarne boje se postavlja na neku nasumičnu vrijednost iz skupa {0, 85, 170, 255} te se nasumično jedna od njih postavlja na 0. Generirane boje su boje iz prethodne dvije težine i sve boje između njih.
4. Elementarne boje se generiraju na isti način kao u 2. težini uz to da se treća nasumično izabrana boja ne postavlja na 0 nego na nasumičnu vrijednost iz skupa {0, 85, 170, 255}.
5. U posljednjoj težini svaka od tri elementarne boje poprima nasumične vrijednosti iz skupa {0, 85, 170, 255}.

```
ColorRGB generateColor()
    {
        float[] color = new float[3];
        int generatorColorComplexity = currentColorComplexity + Random.Range(0, Mathf.CeilToInt(cur-
rentColorComplexity / 2f));
        if (generatorColorComplexity < 180)
        {//1. difficulty
            color[0] = 0;
            color[1] = 0;
            color[2] = 0;
            color[Random.Range(0, 3)] = Random.Range(1, 4) * 85f;
        }
        else if (generatorColorComplexity < 360)
        {//2. difficulty
            float randomColor = Random.Range(1, 4) * 85f;
            color[0] = randomColor;
            color[1] = randomColor;
            color[2] = randomColor;
            color[Random.Range(0, 3)] = 0;
        }
        else if (generatorColorComplexity < 720)
        {//3. difficulty
            color[0] = Random.Range(0, 4) * 85f;
            color[1] = Random.Range(0, 4) * 85f;
            color[2] = Random.Range(0, 4) * 85f;
            color[Random.Range(0, 3)] = 0;
        }
        else if (generatorColorComplexity < 1440)
        {//4. difficulty
            float randomColor = Random.Range(0, 4) * 85f;
            color[0] = randomColor;
            color[1] = randomColor;
            color[2] = randomColor;
            color[Random.Range(0, 3)] = Random.Range(0, 4) * 85f;
        }
        else
        {//5. difficulty
            color[0] = Random.Range(0, 4) * 85f;
            color[1] = Random.Range(0, 4) * 85f;
            color[2] = Random.Range(0, 4) * 85f;
        }

        return new ColorRGB(color[0], color[1], color[2]);
    }
```

Za svaku od pet težina odabrana je granica kompleksnosti boja nakon koje nastupa slijedeća težina, u kodu možemo vidjeti da 1. težina traje od 0 do 179 kompleksnosti, 2. težina 180-359, 3. težina 360-719, 4. težina 720-1439 te 5. težina 1440 i iznad.

Jedini problem s ovakvim sustavom su tranzicije između težina, sa prelaskom currentColorComplexity-a sa 179 na 180 odjednom će boje biti potpuno drugačije što će igrač odmah primijetiti. Problem se rješava na način da se granice ne uspoređuju direktno sa vrijednosti colorComplexity-a nego sa sumom njega i nasumično izabranog broja iz nekog raspona.

Na primjer, recimo da je currentColorComplexity = 130, te se on zbraja sa nasumično izabranom vrijednošću u rasponu [0,60]. O ovom slučaju vjerojatnost da će neprijatelj biti boje 2. težine je 16.6%. S vremenom te povećanjem vrijednosti currentColorComplexity ta vjerojatnost se povećava dok ne dosegne vrijednost 180 i težina je potpuno prešla na 2. razinu.

4.7 Neprijatelji

4.7.1 Cube

Najjednostavniji neprijatelj, kocka s generiranom bojom koju igrač mora pogoditi s metkom iste boje da bi ju uništio.

Pri inicijalizaciji Cube-a on si postavlja brzinu kretanja, boju tijela te dohvaća sve AudioSource-ve koje sadrži. AudioSource-vi su indeksirani u polju po redu po kojemu se pojavljuju u inspektoru.

```
void Start()
{
    gameObject.GetComponent<Rigidbody>().velocity = new Vector3(0, 0, -1 * enemyMovementSpeed);
    gameObject.GetComponent<Renderer>().material.color = enemyColor.getColor();
    incorrectHits = 0;
    audioSources = gameObject.GetComponents<AudioSource>();
    correct = audioSources[0];
    incorrect = audioSources[1];
    enemyDeath = audioSources[2];
}
```

Pri točnom pogotku neprijatelja dodaju se bodovi o kojima vodi računa manager scene(u ovoj sceni „GameManager“) te se šalje poruka EnemySpawneru da poveća killStreak nakon čega se neprijatelj uništi.

```
public void Correct()
{
    GameObject.Find("GameManager").GetComponent<GameManager>().increaseScore(scoreValue);
    EnemySpawner.GetComponent<EnemySpawner>().CorrectHit();
    enemyDeath.Play();
    Die();
}
```

Ukoliko igrač dovoljno mnogo puta netočno pogodi istog neprijatelja njegova bodovna vrijednost opada, ovo onemogućuje igraču da sakupi bodove nausmičnim pogađanjem.

```
public void Incorrect()
{
    incorrectHits++;
    if (incorrectHits > 2)
    {
        scoreValue /= 2;
        scoreValue++;
    }
    incorrect.Play();
    EnemySpawner.GetComponent<EnemySpawner>().IncorrectHit();
}
```

U Die() metodi pomoću koje neprijatelj umire ne uništavamo objekt trenutno nego nakon 4 sekunde, a za vrijeme te 4 sekunde su onemogućeni render, kolizije i fizika Cube-a. Razlog tome je to što pri umiranju neprijatelj ispušta sound effect koji bi se prekinuo ako bi smo uništili neprijatelja pošto bi i AudioSource komponenta koja pokreće taj sound effect također bila uništena.

```
void Die(){
    EnemySpawner.GetComponent<EnemySpawner>().enemyCount--;
    gameObject.GetComponent<Rigidbody>().detectCollisions = false;
    gameObject.GetComponent<Rigidbody>().isKinematic = false;
    gameObject.GetComponent<Renderer>().enabled = false;
    DestroyObject(gameObject, 4f);
}
```

U slučaju da se neprijatelj spusti ispod ekrana ili dotakne igrača poziva se Attack() metoda. Igrač gubi jedan život i neprijatelj umire.

```
public void Attack()
{
    GameObject.Find("GameManager").GetComponent<GameManager>().increasePlayerLives(-1);
    incorrect.Play();
    Die();
}
```

4.7.2 Splitter

Složeniji neprijatelj, oblika trostrane prizme s generiranom bojom. Igrač uništava Splitter-a na način da ga za svaku ne-nul elementarnu boju pogodi sa metkom elementarne boje čija vrijednost odgovora jednoj od vrijednosti elementarnih boja Splitter-a. Nakon točnog pogotka ta elementarna boja nestaje sa njega. Kada se na Splitteru uništi poslijednja elementarna boja on umire.

Skripta koju splitter koristi je iznimno slična skripti Cube-a, Ima svoje Correct(), Incorrect(), Die() i Attack() metode.

4.7.3 Cube boss

Malo veći od običnog Cube-a, nakon što se boss-a pogodi sa točnom bojom on ne umire već se smanji i promijeni boju. Nakon 4 točna pogotka on biva uništen.

```
    void Start()
    {
        gameObject.GetComponent<Rigidbody>().velocity = new Vector3(0, 0, -1 * enemyMovementSpeed);
        enemyColor = JuggernautColors[colorIndex];
        gameObject.GetComponent<Renderer>().material.color = enemyColor.getColor();
        gameObject.transform.localScale = new Vector3(2f + colorIndex * 0.2f, 2f + colorIndex * 0.2f, 2f +
colorIndex * 0.2f);

        incorrectHits = 0;
        audioSources = gameObject.GetComponents<AudioSource>();
        correct = audioSources[0];
        incorrect = audioSources[1];
        enemyDeath = audioSources[2];
    }
    public void Correct()
    {
        if (colorIndex == 0)
        {
            GameObject.Find("GameManager").GetComponent<GameManager>().increasePlayerLives(1);
            GameObject.Find("GameManager").GetComponent<GameManager>().increaseScore(scoreValue);
            EnemySpawner.GetComponent<EnemySpawner>().CorrectHit();
            Die();
        }
        else
        {
            EnemySpawner.GetComponent<EnemySpawner>().CorrectHit();
            correct.Play();
            colorIndex--;
            enemyColor = JuggernautColors[colorIndex];
            gameObject.GetComponent<Renderer>().material.color = enemyColor.getColor();
            gameObject.transform.localScale = new Vector3(2f + colorIndex * 0.2f, 2f + colorIndex * 0.2f,
2f + colorIndex * 0.2f);
        }
    }
```

Cube boss(skraćeno CB) se razlikuje od običnog cube-a po tome što sadrži više boja koje treba pogoditi. To je ostvareno na slijedeći način.

Unutar EnemySpawner objekta, osim što instancira CB on mu također prosljeđuje polje boja(JuggernautColors) koje on poprima kao i indeks posljednje boje u polju(colorIndex).

Pri točnom pogotku indeks se smanjuje za jedan i primjenjuje se nova boja te se CB smanjuje u veličini.To ponavlja dok mu se ne pogodi posljednja boja nakon čega umire i daje igraču jedan dodatan život.

4.7.4 Kolizije

Procesiranje kolizija između metka i neprijatelja se vrši na metku unutar komponente BulletBehaviour.cs. U produžetku se nalazi mali isječak koda.

```
    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.name == "enemy(Clone)")
        {
            ColorRGB enemyColor = other.GetComponent<EnemyBehaviour>().enemyColor;
            if (bulletColor == enemyColor)
            {
                other.GetComponent<EnemyBehaviour>().Correct();
                RGBMeterBehaviour.ClearAllHints();
            }
            else
            {
                other.GetComponent<EnemyBehaviour>().Incorrect();
                if (other.GetComponent<EnemyBehaviour>().incorrectHits > incorrectHitsForHint)
                {
                    RGBMeterBehaviour.UpdateAllHints(bulletColor.r - enemyColor.r, bulletColor.g -
enemyColor.g, bulletColor.b - enemyColor.b);
                }
            }

            Hit();
        }
    }
```

Metoda OnTriggerEnter se automatski poziva svaki puta kada Collider gameObject-a na kojem se skripta nalazi sudari sa nekim drugim objektom koji također ima Collider. Kao argument metode OnTriggerEvent prosljeđuje referenca na drugi gameObject.

Pošto metoda se poziva pri svakoj koliziji potrebno je prvo odrediti je li drugi objekt onaj za koji su nam važne kolizije. Nakon što smo utvrdili da je drugi objekt tipa neprijatelj provjeravamo da li boja metka odgovara boji neprijatelja, ukoliko jest poziva se metoda Correct() na neprijateljskom objektu, inače se poziva metoda Incorrect(). Ako je igrač previše puta pogodio neprijatelja krivom bojom na UI-u RGBmeter-a se prikazuju hintovi koliko koje boje treba više ili manje.

5 Zaključak

Izrada videoigara je težak i složen posao za koji su se tek nedavno počele standardizirati procedure i metodologije razvoja. Činjenica da je na projektu potrebno mnogo osoblja različitih disciplina samo dalje otežava koordinaciju projekta.

Igra Chroma u svom trenutnom stanju je igriva, moguće ju je zvati gotovim proizvodom ali joj nedostaje dorada detalja, unaprjeđenje dizajna i optimizacija koda .

U novoj verziji, igra se planira doraditi u nekoliko segmenata. Vizualno će biti ljepša, a neprijatelji će imati nekakvu animaciju umiranja umjesto da samo nestanu, a user interface-u će biti poboljšan vizualni dizajn pošto trenutno većina meni-a izgleda pusto.

Sa programske strane kod će se bolje strukturirati a pristup kodiranju bolje isplanirati. Dobar dio vremena razvoja je proveden refaktoriranjem postojećeg koda radi očuvanja čitljivosti i preglednosti. Npr. u trenutnoj verziji igre unutar direktorija „ChromaSource.zip\PH2\Assets\Game\player“ datoteka „BulletBehaviour.cs“ sadrži ne samo kod za detekciju kolizije metka sa neprijateljima nego također rukovodi uspoređivanje boja sa neprijateljima. Taj dio koda će biti premješten u skripte pojedinih neprijatelja nakon čega će metci pri koliziji samo poslati poruku neprijateljima da su pogođeni određenom bojom.

Sa stajališta dizajnera igre ostaje podešavanje težine, što iziskuje mnogo vremena. U razmatranju je i mode gdje se koristi suptraktivni CMYK model boja, više tipova neprijatelja, power-up-ovi i sl.

Stoga, iako je razvoj igre završio, igra nije i nikada neće biti gotova jer uvijek postoji područje u kojem se ona može poboljšati i razvijati.

6 Prilozi

Uz ovaj rad priložene su slijedeće datoteke:

- Chroma.zip (sadrži igru Chroma)
- ChromaSource.zip (sadrži source projekt igre Chroma)

7 Popis slika

8 Literatura

1980s in video gaming - Wikipedia. (n.d.). Preuzeto 8. 9 2018 iz Wikipedia: https://en.wikipedia.org/wiki/1980s_in_video_gaming

1990s in video gaming - Wikipedia. (n.d.). Preuzeto 8. 9 2018 iz Wikipedia: https://en.wikipedia.org/wiki/1990s_in_video_gaming

Independent video game development - Wikipedia. (n.d.). Preuzeto 8. 9 2018 iz Wikipedia: https://en.wikipedia.org/wiki/Independent_video_game_development

RGB color model - Wikipedia. (n.d.). Preuzeto 28. 8 2018 iz Wikipedia: https://en.wikipedia.org/wiki/RGB_color_model

Unity - Manual : Unity User Manual. (n.d.). Preuzeto 26. 8 2018 iz Unity: https://docs.unity3d.com/Manual/

Unity - Scripting API. (n.d.). Preuzeto 26. 8 2018 iz Unity: https://docs.unity3d.com/ScriptReference/

Video game industry - Wikipedia. (n.d.). Preuzeto 8. 9 2018 iz Wikipedia: https://en.wikipedia.org/wiki/Video_game_industry